A First Course on Kinetics and Reaction Engineering

Example S2.1

Problem Purpose

This example illustrates the use of the MATLAB template file SolvNonDif.m to solve a set of non-linear equations.

Problem Statement

In equations (1) through (4) A, B and C are constants with values of 0.2083 mol/min, 5.472 min/mol and 0.4164 mol/min, respectively. Solve the equations for z_1, z_2, z_3 and z_4, and then compute the ratio of z_1 to z_2.

$$f_1(\underline{z}) = A - Bz_1^{1.5}z_2^{0.5} - z_1 = 0 \tag{1}$$

$$f_2(\underline{z}) = C - Bz_1^{1.5}z_2^{0.5} - z_2 = 0 \tag{2}$$

$$f_3(\underline{z}) = Bz_1^{1.5}z_2^{0.5} - z_3 = 0 \tag{3}$$

$$f_4(\underline{z}) = Bz_1^{1.5}z_2^{0.5} - z_4 = 0 \tag{4}$$

Problem Analysis

There are four equations to be solved for the values of four unknowns. The equations do not contain derivatives or integrals and can be written in the vector form of equation (5). Therefore, the MATLAB template file SolvNonDif.m can be used to solve them numerically.

$$0 = \underline{f}(\underline{z}) \tag{5}$$

Problem Solution

The solution presented here will follow the step-by-step instructions provided with Supplemental Unit S2 for using SolvNonDif.m. To begin, a copy of SolvNonDif.m was saved as S2_Example_1.m, and all modifications were made to that copy. The long multi-line comment at the top of the file was replaced with a comment describing the purpose of the modified template file. The function statement that follows then was changed to match the new filename. The first required modification follows immediately after the function statement, and it involves entering values for all constants that are involved in the problem. Here the only constants are A, B and C, and they are already in consistent units of moles and minutes. Following all these changes, the top of the modified file looks as shown in Listing 1.

```
% Modified version of the MATLAB template file SolvNonDif.m used to solve
% Example 1 of Supplemental Unit S2 of "A First Course on Kinetics and
% Reaction Engineering."
%
function z = S2_Example_1
    % Known quantities and constants (in units of mol and min)
    A = 0.2083;
    B = 5.472;
    C = 0.4167;
```

Listing 1. Modified portion of SolvNonDif.m after renaming and making the first required modification.

The second required modification appears within the internal function `nlaeqns`. It involves adding code to evaluate the functions f_1 through f_4 in equations (1) through (4) as the elements of the column vector `f`. This modification is straightforward, and the results are shown in Listing 2. Notice that the constants entered previously are available at this point in the program. Notice also that the value of the first function is calculated and saved as the first row of the vector `f`, the value of the second function is calculate and saved as the second row, and so on.

```
    % Function that evaluates the equations
    function f = evalEqns(z)
        term1 = z(1)^1.5*z(2)^0.5;
        f = [
            A - B*term1 - z(1);
            C - B*term1 - z(2);
            B*term1 - z(3);
            B*term1 - z(4);
        ];
    end % of internal function evalEqns
```

Listing 2. Modified version of the internal function nlaeqns.

The third modification occurs in the main function of S2_Example_1, and it involves entering guesses for the unknowns. Here I have no physical basis for making any guesses, so I simply used a value of 1.0 for each guess as shown in Listing 3. Notice again, the guess for z_1 is entered as the first row of the vector `z_guess`, the guess for z_2 is entered as the second row, and so on.

```
    % guesses for the solution
    z_guess = [
        1
        1
        1
        1
    ];
```

Listing 3. The third required modification to provide guesses for the unknowns.

The final required modification appears after the equations have been solved. It involves inserting the code to calculate any additional quantities that are needed whose values depend upon the solution to the equations. Here we are asked to calculate the ratio of z_1 to z_2. Listing 4 shows this modification.

```
    % compute the requested ratio
    ratio = z(1)/z(2)
```

Listing 4. The final required modification where additional quantities are calculated using the results from solving the equations.

That completes all the required modification, so S2_Example_1.m was saved to preserve the modifications. A copy of S2_Example_1.m is provided with this supplemental unit. The template file does not require any input arguments, so it is ready to be executed. This was done by typing the first line shown in Listing 5; the resulting output is also shown in Listing 5.

```
>> z = S2_Example_1;

Equation solved.

fsolve completed because the vector of function values is near zero
as measured by the default value of the function tolerance, and
the problem appears regular as measured by the gradient.

<stopping criteria details>

The solver found the following values for the unknowns:

z =
    0.1047
    0.3131
    0.1036
    0.1036

The corresponding values of the functions being solved are as follows:

f =
   1.0e-11 *

   -0.1535
   -0.1532
    0.1535
    0.1535


ratio =
    0.3343
```

Listing 5. Output generated upon execution of the modified template file, S2_Example_1.m

The first part of the output is a message generated by the built-in MATLAB function `fsolve`. The message states that the equations were solved successfully, noting that the iterative solution process (see the informational reading for this supplemental unit) terminated because all of the functions were close to zero.

The next output items are the values of z_1, z_2, z_3 and z_4 as the column vector z, and the values of the functions f_1 through f_4 evaluated at those values of z_1, z_2, z_3 and z_4. We can see that they are all of the order of 10^{-11}, which is sufficiently close to zero for this problem.

The ratio of z_1 to z_2 is listed next. This output was generated by the code we added during the fourth required modification of the template file.

Note that the column vector z was also returned by the template file, but because the function call in Listing 5 ends with a semicolon, the values were not printed a second time. Nonetheless, because z was returned, it can be used in additional calculations entered at the MATLAB command prompt. In contrast, the value of ratio was not returned by the template file, it was only printed from within the function S2_Example_1. If we tried to perform a calculation at the MATLAB command prompt using ratio, MATLAB would not know the value of ratio. This can be seen in Listing 6 where commands to double `z(1)` and to double `ratio` were entered at the MATLAB command prompt after S2_Example_1 had executed. The first calculation succeeds because z was returned by S2_Example_1 and so is defined, but the second fails because `ratio` was not returned, only printed. (It would be easy to modify the template file so it returned `ratio` as well as z. Alternatively, since z is returned, one could calculate `ratio` at the MATLAB command prompt after S2_Example_1 returned the values of z instead of calculating `ratio` within the template file.)

```
>> double_z1 = 2*z(1)

double_z1 =

    0.2093

>> double_ratio = 2*ratio
Undefined function or variable 'ratio'.
```

Listing 6. Results of attempting to use returned results and printed results from S2_Example_1.